LABORATORY WORK № 2.1.

RECURSIVE ALGORITHMS

**The purpose of the laboratory work**

The purpose of laboratory work № 2.1 is to master the theoretical material and gain practical experience in creating recursive algorithms and writing appropriate programs.

**Formulation of the problem**

Given a natural number *n*. Find the sum of the first *n* terms of a series of numbers given by a recurrent formula. Solve the problem in three ways (write three programs): 1) in the program use a recursive procedure or function that performs the calculation of both members of the series, and the amount on the recursive descent (nontail recursion); 2) in the program to use a recursive procedure or function that performs the calculation of both members of the series, and the amount of recursive return (tail recursion); 3) in the program to use a recursive procedure or function that performs the calculation of the members of the series on the recursive descent, and the calculation of the amount on the recursive return (combined recursion).

Programs must work correctly for any natural *n* including *n* = 1.

**Contents of the report**

1. General problem statement and tasks for a specific variant.

2. The text of all three programs.

3. Program testing. For the purpose of testing it is necessary to write a cyclic variant of the program (for loop), and also to carry out the calculation of the formula on the calculator.

4. As a result, print the test data (for *n* = 5) as a cyclic program and calculations on a calculator, and solve the problem with all three recursive programs. The results of the calculations must match in all ways.

**Test questions**

1. Definition of a recursive object.

2. Determining the depth and current level of the recursion.

3. Form of performing recursive actions on a recursive descent (nontail recursion).

4. Form of performing recursive actions on recursive return (tail recursion).

5. Form of performing recursive actions on both recursive descent and recursive return.

**Variants for individual tasks**

**Variant** № 1

*F*1 = 1; *F*2 = −*x/*2; *Fi* = −*Fi −*1⋅*x⋅*(2*i −*3)/(2*i*), *i* > 2;

∑ *Fi*  = , |*x*| < 1.

**Variant** № 2

*F*1 = 1; *F*2 = −*x/*3; *Fi* = −*Fi −*1⋅*x⋅*(3*i −*7)/(3*i−*3), *i* > 2;

∑ *Fi*  = , |*x*| < 1.

**Variant** № 3

*F*1 = 1; *Fi* = −*Fi −*1⋅*x⋅*(2*i −*3)/(2*i−*2), *i* > 1;

∑ *Fi*  = 1/, |*x*| < 1.

**Variant** № 4

*F*1 = 1; *Fi* = −*Fi −*1⋅*x⋅*(3*i −*5)/(3*i−*3), *i* > 1;

∑ *Fi*  = 1/, |*x*| < 1.

**Variant** № 5

*F*1 = 1; *Fi* = *Fi −*1⋅(2*x/*3 – 1); *i* > 1;

∑ *Fi*  = 1.5/*x*, 1 < *x* < 2.

**Variant** № 6

*F*1 = 1; *Fi* = *Fi −*1⋅(*i −*1)⋅1*x ⋅*ln2/*i*; *i* > 1;

∑ *Fi*  = 2*x*.

**Variant** № 7

*F*1 = (*x −*1)/(*x +* 1); *Fi+*1= *Fi*⋅ (2*i−*1)(*x −*1)2/((2*i+*1)⋅(*x +* 1)2); *i* > 1;

∑ *Fi*  = − 0.5ln *x*; *x* < 1.

**Variant** № 8

*F*1= *x −* 1; *Fi+*1= −*Fi* ⋅ (*x –* 1)⋅*i*/(*i +* 1); *i* > 0;

∑ *Fi*  = ln *x*, 0 < *x* < 2;

**Variant** № 9

*F*1 = (*x −*1)/*x*; *Fi+*1= *Fi*⋅ *i*⋅(*x −*1)/(*i⋅x + x*); *i* > 1;

∑ *Fi*  = ln *x*; *x* >0.5.

**Variant** № 10

*F*1 = *x*; *Fi* = −*Fi −*1⋅ *x*⋅(*i −*1)⋅ (*i −*2)/(*i*2 *−i*); *i* > 1;

(1+*x*)∑ *Fi*  = ln (1+*x*); −1<*x* <2.

**Variant** № 11

*F*1= *x*; *Fi+*1= −*Fi* ⋅ *x*2/(4*i*2 *+* 2*i*); *i* > 0;

∑ *Fi*  = sin *x*.

**Variant** № 12

*F*1 = 1; *Fi+*1= −*Fi* ⋅ *x*2/(4*i*2*−* 2*i*); *i* > 0;

∑ *Fi*  = cos *x*.

**Variant** № 13

*F*1= *x*; *Fi+*1= *Fi* ⋅ *x*2 (2*i −*1)2/(4*i*2+ 2*i*); *i* > 0;

∑ *Fi*  = arsin *x, −*1< *x*< 1.

**Variant** № 14

*F*0 = *x*; *Fi* = *Fi −*1⋅ (2*i* −1)2⋅*x*2/(2*i*(2*i*+1)); *i* > 0;

π/2 − ∑ *Fi*  = arccos *x*.

**Variant** № 15

*F*1= *x*; *Fi+*1= −*Fi* ⋅ *x*2 (2*i −*1)/(2*i* + 1); *i* > 0;

∑ *Fi*  = arctg *x*, |*x*| <1;

**Variant** № 16

*F*1 = *x*; *Fi +*1= *Fi* ⋅ (2*i* −1)2 ⋅*x*2/(4*i*2 +2*i*); *i* > 0;

∑ *Fi*  = arsin *x*, |*x*| <1;

**Variant** № 17

*F*1 = *x*; *Fi+*1= *Fi*⋅ *x*2/(4*i*2+2*i*); *i* > 0;

∑ *Fi*  = sh *x*, |*x*| <106;

**Variant** № 18

*F*1 = 1; *Fi+*1= *Fi*⋅ *x*2/(4*i*2 −2*i*); *i* > 1;

∑ *Fi*  = ch *x*, |*x*| <106;

**Variant** № 19

*F*1= *x*; *Fi+*1= −*Fi* ⋅ *x*2(2*i–* 1)2/(4*i*2 *+* 2*i*); *i* > 0;

∑ *Fi*  = arcsh *x*, |*x*| <1;

**Variant** № 20

*F*1= *x*; *Fi+*1= *Fi* ⋅ *x*2(2*i−*1)/(2*i+*1); *i* > 0;

∑ *Fi*  = arcth *x*, |*x*| <1;

**Variant** № 21

*F*0= 4/3; *F*1= *x/*2; *Fi+*1= *−Fi* ⋅ *x* (2*i−*1)/2*i*; *i* > 1;

∑ *Fi*  = , |*x*| <1;

**Variant** № 22

*F*1= *x –* 1; *Fi+*1= *−Fi* ⋅ (*i −*1)(*x* – 1)/*i*; *i* > 0;

∑ *Fi*  = ln *x*, 0 < *x* <2;

**Variant** № 23

*F*1= (*x –* 1)/*x*; *Fi+*1= *Fi* ⋅ (*i −*1)(*x* – 1)/(*ix*); *i* > 0;

∑ *Fi*  = ln *x*, 0.5 < *x*;

**Variant** № 24

*F*0= 1; *Fi+*1= −*Fi* ⋅ *x*2/*i*; *i* > 0;

∑ *Fi*  = *e*−*x⋅x* ;

**Variant** № 25

*F*1= *x*; *Fi* = −*Fi−*1⋅ *x*(*i* – 1)/*i*; *i* > 1;

∑ *Fi*  = ln (1 + *x*), −1 < *x* < 1;

**Variant** № 26

*F*1= *x*; *Fi* = *Fi−*1⋅ *x*2(2*i* – 3)/(2*i−*1); *i* > 1;

∑ *Fi*  = 0.5ln ((1 + *x*)/(1 – *x*)), −1 < *x* < 1;

LABORATORY EXERCISE №2.2.

Linked DYNAMIC DATA STRUCTURES. LISTS

**The goal of the laboratory work**

The goalof the laboratory work №2.2 is to master the theoretical material and gain practical experience in using linked dynamic data structures in the form of single- and double-linked lists in the design different algorithms.

**Formulation of the problem**

1. Create a list of *n* (*n*> 0) elements (*n* is entered from the keyboard), if the other number of elements is not specified in a specific task.

2. The type of keys (information fields) is set by the variant.

3. Take the values ​​of the list elements independently so that you can demonstrate the correct operation of the program algorithm. The student can enter the values ​​of the list items in any way (random numbers, generate values ​​by a formula, input from a file or from the keyboard).

4. The type of the list (queue, stack, deck, direct single-linked linear list, inverted single-linked linear list, double-linked linear list, single-linked ring list, double-linked ring list,) is choosen independently for the most appropriate solution of the problem set by the variant.

5. Perform the actions specified in the variant in the created list and correctly free the list memory.

6. When performing the specified actions, outputting the values ​​of the elements and freeing the memory of the list, assume that the length of the list (number of elements *n* or 2*n*) is unknown at the time of these actions.

7. Duplicate parts of the algorithm must be designed in the form of procedures or functions (to create, process, output and free the memory lists) with the transfer of the list using the parameters.

**Contents of the report**

1. General problem statement and tasks for a specific option.

2. Program text.

3. Testing the program, ie, the initial data and the corresponding results.

4. As a result, print out the test data and solve the problem on a computer.

**Control questions**

1. Are there any restrictions on the number of items in the list, if so, which ones are?

2. What are the types of list data structures in terms of their logical use (stack, queue, etc.). Explain their features and differences between them.

3. What are the requirements for the structure of a connected dynamic data element?

4. What is special about describing types to create coherent dynamic data?

5. What should be the structure of a linear double-linked list element?

6. How many pointers and what purpose is needed to work with the queue?

7. How many pointers and what purpose is needed to work with the stack?

8. How many pointers and what purpose is needed to work with the deck?

9. How many pointers and what purpose is needed to work with linear unlinked lists?

10. How many pointers and what purpose are needed to work with linear double-linked lists?

**Task variants**

**Variant 1**

The key of a list element is a string of the length no more than10 symbols, which contains the latin characters. To sort the list elements in the lexicographic order, not to use the additional data structures, except simple variables (ie, «in the same place»), by the method of selection.

**Variant 2**

The keys of the list elements are reals.The number of the list elements has to be equval to 2*n*. To calculate the value of the formula:

( *a*1 − *a*2n ) ( *a*3 − *a*2n−2 ) …( *a*2n−1 − *a*2 ) ,

where *a*i is the i-th list element.

**Variant 3**

The key of a list element is a string of latin characters and digits. Reorder the list in such a way, that all the digits stay at the beginning of the list, not to use the additional data structures, except the simple variables (ie, «in the same place»).

**Variant 4**

The given list is a queue. The key of a list element is a symbol. To rewrite the elements of this queue to another queue in the inverse order, not to count the number of elements in the queue.

**Variant 5**

The key of a list element is the real number. To perform the cyclic shift of the list elements to k positions left (k is the natural number and it is not higher than the number of of the list elements). If necessary it is allowed to use another list, other data structures, except the simple variables.

**Variant 6**

The keys of the list elements are reals. To sort the list elements in the descending order, not to use the additional data structures, except the simple variables (ie, «in the same place»), by the bubble sorting using a flag.

**Variant 7**

The key of a list element is an integer. The number elements of the list has to be equval to 2*n*. To calculate the value of the formula:

*a*1*a*2n + *a*2*a*2n−1 + … + *a*n*a*n+1 ,

where *ai* –is the *i*-th list element.

**Variant 8**

The key of a list element is the nonzero integer number, and the number negative numbers is equal to the number of positive ones. Reorder the list so that deriving a sequence of numbers with the alternating signs, not to use the additional data structures, except the simple variables (ie, «in the same place»).

**Variant 9**

The keys of the list elements are different reals. To find the maximum and minimum list elements. To insert to the list two new elements: after the element with the maximum value insert the element with the minimum value, and after the element with the minimum value insert the element with the maximum value.

**Variant 10**

The key of a list element is an integer. To derive the number of the list elements, which value яких is higher than given integer М, and to insert new elements after the *k*-th element of the list (*k* is the natural number which is not higher the number of the list elements).

**Variant 11**

The key of a list element is the Latin character. To sort the list elements in the lexicographic order, not to use the additional data structures, except simple variables (ie, «in the same place»), by the method of select insertion.

**Variant 12**

The key of a list element is the integer. To calculate the value of the formula:

( *a*1 + *a*2 + 2*a*n ) (*a*1 + *a*2 + 2*a*n−1 )…( *an−*1 + *a*n + 2*a*2) ,

where *a*i is the *i*-th list element.

**Variant 13**

The keys of the list elements are integers. To reorder the list elements so that firstly the positive are placed, then zero are, and negative elements after them, not to use the additional data structures, except simple variables (ie, «in the same place»).

**Variant 14**

The key of a list element is a string of the length no more than 5 symbols. Reorder the list so that the list elements are placed in the negative order (to perform the « mirrored order» of the list), not to use the additional data structures, except simple variables (ie, «in the same place»).

**Variant 15**

The key of a list element is the real number. Reorder the elements of the list in such a way, that its elements are placed in the following order:

*a*1, *a*n, *a*2, *a*n−1, …, *a* [(n+1)/2],

where *a*i is the *i*-th list element. If necessary it is allowed to use the another list, otherdata structures, except simple variables.

**Variant 16**

The key of a list element is the real number. To expand the list, adding to its end the same elements, but in the negative order, not to use the additional data structures, except simple variables (ie, «in the same place»).

**Variant 17**

The key of a list element isare the real number. To calculate значення of the formula:

*a*1*a*n + *a*2*a*n−1 + … + *a*n*a*1 ,

where *a*i is the *i*-th list element.

**Variant 18**

The key of a list element is the nonzero integer. They are placed in the following order: 10 positive ones, 10 negative and so on.

The number of the list elements *n* has to be a multiple of 20. Reorder the list so that placing of the elements is as follows: 5 positive ones, 5 negative and so on, not to use the additional data structures, except simple variables (ie, «in the same place»).

**Variant 19**

Two lists are given, the list **S1** of the length 2*n* elements and the list **S2** of the length *n* elements. The keys elements of both lists are naturals. To insert the list **S2** in the middle of the list **S1**, not to use the additional data structures, except simple variables (ie, «in the same place»).

**Variant 20**

The key of a list element is the integer. The number elements of the list has to be equval to 2*n*. Reorder the elements of the list so placing of the elements is as follows:

*a*1, *a*n+1, *a*2, *a*n+2, …, *a*n, *a*2n,

where *a*i is the *i*-th list element, not to use the additional data structures, except simple variables (ie, «in the same place»).

**Variant 21**

The key of a list element is the real number. To perform the following operations: if the list elements are ordered in the descending order, then remain it without exchanges, otherwise to rewrite the elements of the list in the descending order. If necessary it is allowed to use the additional list, other data structures, except simple variables.

**Variant 22**

The key of a list element is the nonzero integer. They are placed in the following order: 5 negative, 5 positive ones and so on. The number of the list elements *n* has to be a multiple of 20. Reorder the list elements so that placing of the elements is as follows: 10 negative, 10 positive ones and so on, not to use the additional data structures, except simple variables (ie, «in the same place»).

**Variant 23**

The key of a list element is an integer. To perform the cyclic shift of the list elements to *k* positions right (*k*  is natural and is higher than the number of the list elements). If necessary, it is allowed to use another list, other data structures, except simple variables.

**Variant 24**

The key of a list element is the nonzero integer. The number of the list elements *n* has to be a multiple of 4, and the number of negative numbers is equal to the number of positive ones. Reorder the list elements so placing of the elements is as follows: two positive ones, two negative and so on, not to use the additional data structures, except simple variables (ie, «in the same place»).

**Variant 25**

The key of a list element is the Latin character. Reorder the list so that firstly the elements with the loud Latin characters are placed, а then the elements with the consonantal Latin characters, without changing the initial mutual order of the characters. For example:

initial list: university

the result: uieiynvrst

If necessary it is allowed to use the other list, other data structures, except simple variables.

**Variant 26**

The key of a list element is the natural. Reorder the list so that firstly the elements which are multiple of 3, then the elements which are multiple of 3 plus 1, and finally, the rest of elements, not to use the additional data structures, except simple variables (ie, «in the same place»).